Cray Scientific Libraries

Keita Teranishi

Scientific Libraries Group

Cray Scientific Libraries

LibSci	PETSc	FFT	ACML
ScaLAPACK	PETSc	FFTW	FFT
BLAS (libGoto)	HYPRE	SPIRAL	RNG
LAPACK	ParMETIS		
IRT	MUMPS		
CRAFFT	SuperLU		
	SuperLU_dist		
	CASK		

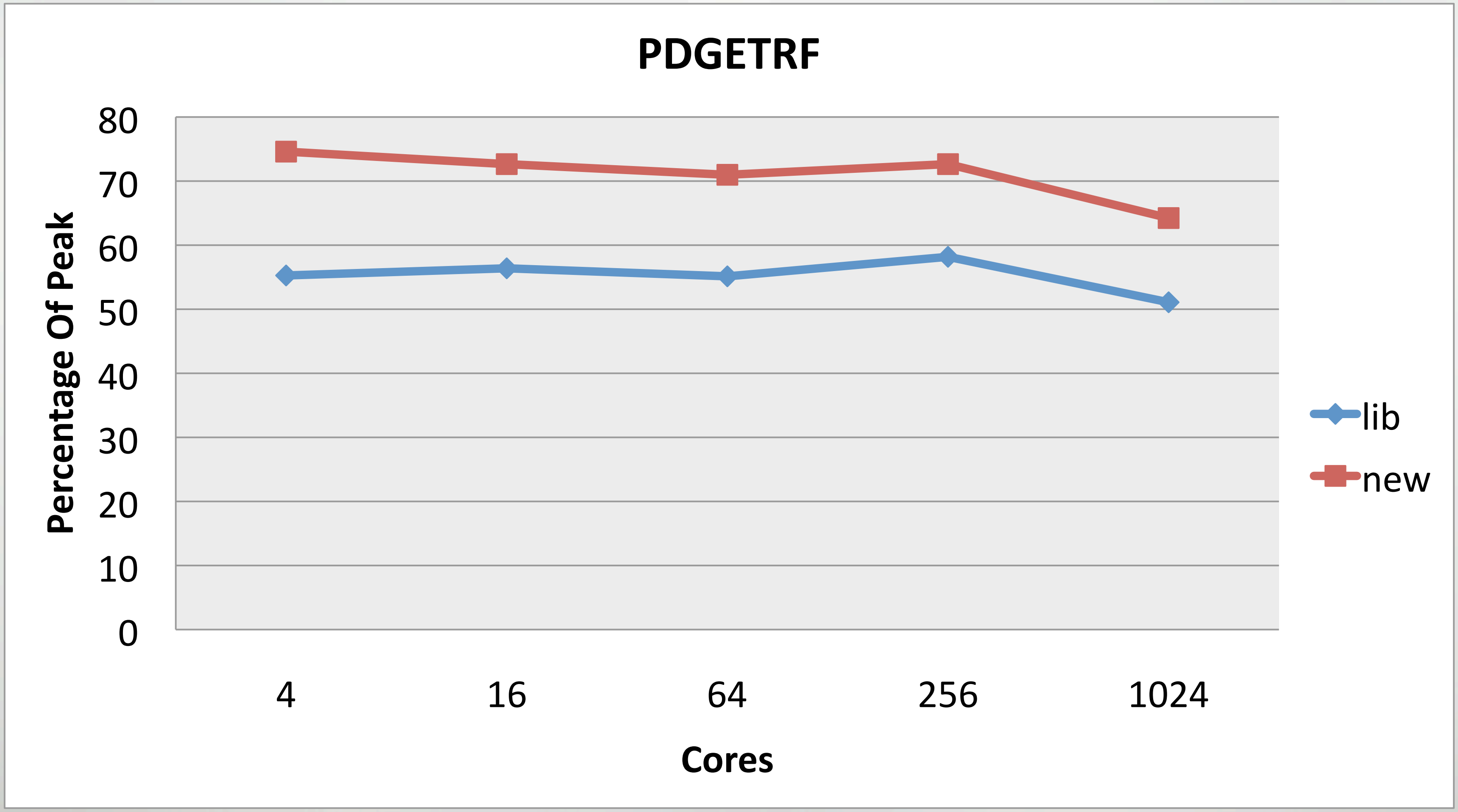
Scientific Libraries Recent Work

- Released libsci 10.3.3 – March 2009
  - libgoto 1.29 with performance improvements to BLAS and LAPACK
  - CRAFFT 1.1 with Single precision
- Libsci 10.3.4 – April 2009
  - Dynamic libraries (.so files) support
  - Documentation bug fixes
- Released PETSc 3.0.0 – February 2009
  - PETSc + HYPRE, SuperLU, SuperLU_DIST, MUMPS, ParMETIS,
  - CASK-1.0 improves iterative solver performance by 5-50% (depending of problems)
- FFTW version 3.2.1 – April 2009
  - Bug fix

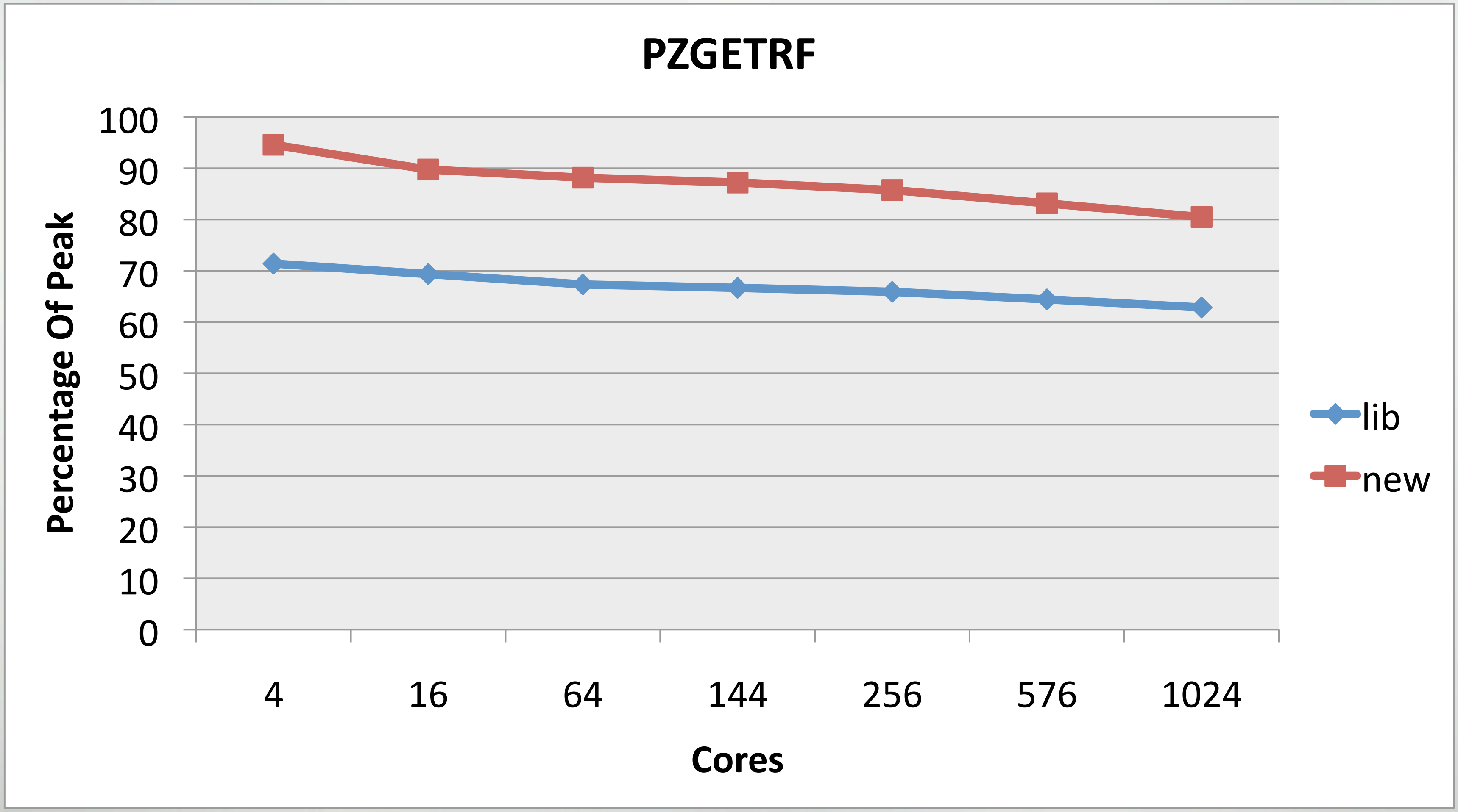
Major Scientific Libraries releases

- libsci-10.4.0 (June 2009)
  - CRAFFT 2.0 includes SPIRAL package support for the underlying computation of the serial FFT
  - LU, QR, Cholesky performance improvement in LAPACK and ScaLAPACK
- CASK-1.1 for Istanbul Processors (June 2009)
- Trilinos-9.0.2 with CASK-2.0 (August 2009)

New ScaLAPACK LU on XT5



New ScaLAPACK LU on XT5

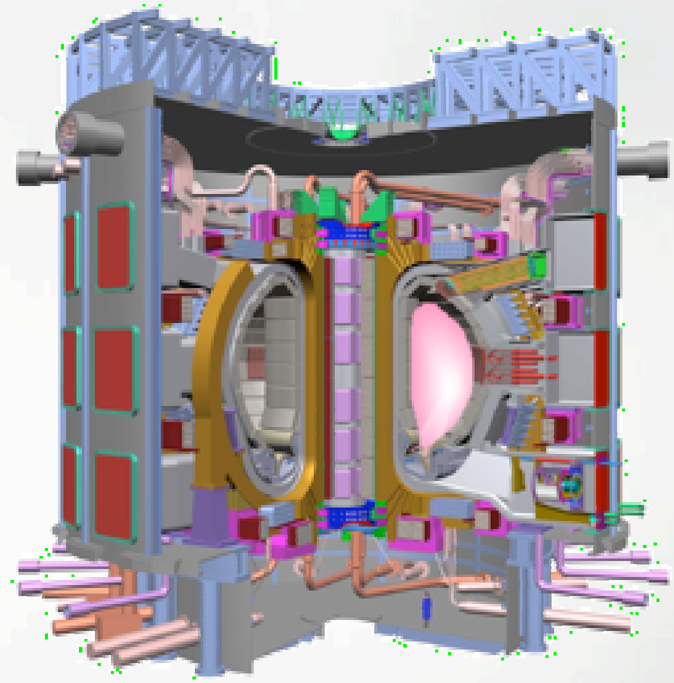


Iterative Refinement Toolkit

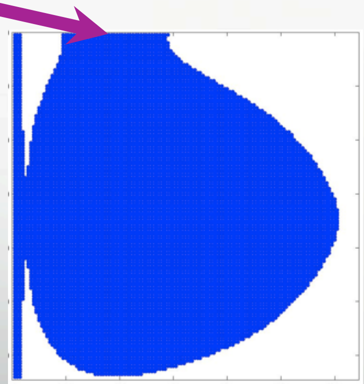
- Solves linear systems in single precision whilst obtaining solutions accurate to double precision
  - For well conditioned problems
- Serial and Parallel versions of LU, Cholesky, and QR
- From LibSci-10.2.0, there are now 2 ways to use the library
  1. **IRT Benchmark routines**
     - Uses IRT 'under-the-covers' without changing your code
       - Simply set an environment variable
       - Useful when you just want a quick-and-dirty factor/solve
  2. **Advanced IRT API**
     - If greater control of the iterative refinement process is required
       - Allows
         - condition number estimation
         - error bounds return
         - minimization of either forward or backward error
         - 'fall back' to full precision if the condition number is too high or IRT fails
         - max number of iterations can be altered by users

Example: AORSA Fusion Energy

- "High Power Electromagnetic Wave Heating in the ITER Burning Plasma"
- rf heating in tokamak
- Maxwell-Bolzmann Eqns
- FFT
- Dense linear system
- Calc Quasi-linear op

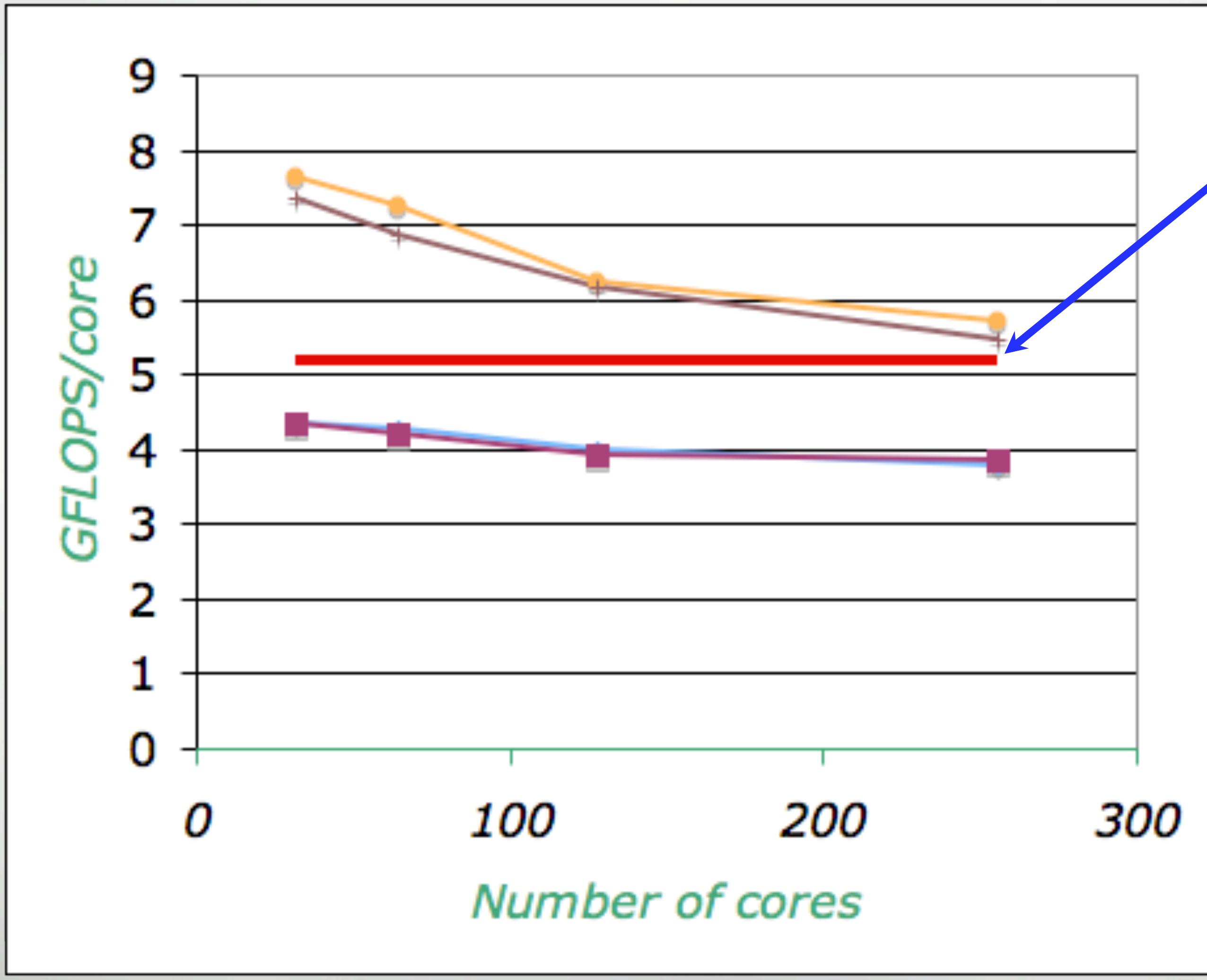


ITER-FEAT



Courtesy
Richard Barrett
Oak Ridge National Laboratory

AORSA solver performance - 128x128 grid



Theoretical Peak

Courtesy
Richard Barrett

Oak Ridge National Laboratory

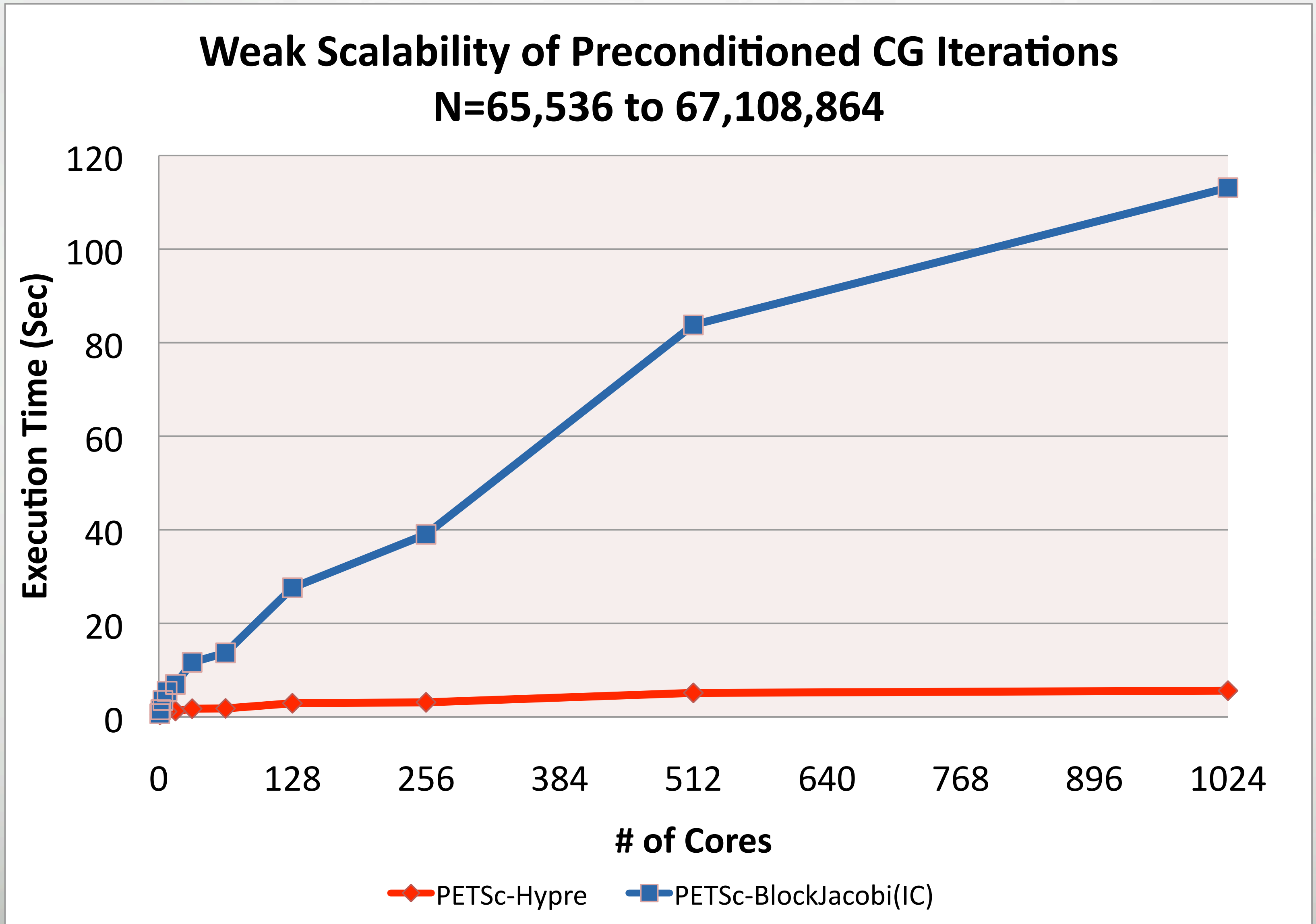
PETSc (Portable, Extensible Toolkit for Scientific Computation)

- Serial and Parallel versions of sparse iterative linear solvers
  - Suites of iterative solvers
    - CG, GMRES, BiCG, QMR, etc.
  - Suites of preconditioning methods
    - IC, ILU, diagonal block (ILU/IC), Additive Schwartz, Jacobi, SOR
  - Support block sparse matrix data format for better performance
  - Interface to external packages (Hypre, SuperLU_DIST,MUMPS)
  - Fortran and C support
  - Newton-type nonlinear solvers
- Large user community
- http://www-unix.mcs.anl.gov/petsc/petsc-as

PETSc External Packages

- Cray provides external scientific computing packages to strengthen the capability of PETSc
  - Hypre: scalable parallel preconditioners
    - AMG (Very scalable and efficient for specific class of problems)
    - 2 different ILUs (General purpose)
    - Sparse Approximate Inverse (General purpose)
  - ParMetis: parallel graph partitioning package
  - MUMPS: parallel multifrontal sparse direct solver
  - SuperLU: sequential left-looking sparse solver
  - SuperLU_DIST: parallel right-looking sparse direct solver with static pivoting

PETSc-Hypre (BoomerAMG) Scalability



Scientific Library Focus in 2009 – Auto-tuning

- **Auto-tuning: automate code generation and a huge number of empirical performance evaluations to configure software to the target platforms.**
  - Cray Adaptive Sparse Kernels (CASK)
  - Cray Adaptive FFT (CRAFFT)
  - More scientific codes will be auto-tuned
- **Adaptivity:** make runtime decisions to choose the best kernel/library/routine
  - Cray Adaptive FFT (CRAFFT)
  - CASK
- **Performance:**
  - Iterative Solver Performance
  - FFT performance
  - Multi-core optimizations (Istanbul Processors)

Cray Adaptive FFT (CRAFFT)

- In FFTs, the relevant problems are
  - Which library choice to use?
  - How to use complicated interfaces (e.g., FFTW)
- Standard FFT practice
  - Do a plan stage
    - Deduced machine and system information and run micro-kernels
    - Select best FFT strategy
  - Do an execute

Our system knowledge can remove some of this cost!

Major problem with FFT libs

- Which library to choose?
  - We want the best possible FFT performance
    - To date, we have seen excellent performance from FFTW
  - Do NOT want to change application code frequently
- How to use the complicated interfaces???
  - FFTW can be really difficult to use
    - E.g., 2d FFT with LDA > size, 14 arguments!!!

```
call dfftw_plan_many_dft(plan,rank,n,howmany, &
          input,inembed, &
           istride,idist, &
          output,onembed, &
           ostride,odist, &
          expon,FFTW_flags)
```

CRAFFT library

- CRAFFT is designed with simple-to-use interfaces
  - Planning and execution stage can be combined into one subroutine call
  - Underneath the interfaces, CRAFFT calls the appropriate FFT kernel

- CRAFFT provides both offline and online tuning
  - Offline tuning
    - Which FFT kernel to use
    - Pre-computed PLANs for common-sized FFT
      - No expensive plan stages
  - Online tuning is performed as necessary at runtime as well

- At runtime, CRAFFT will adaptively select the best FFT kernel to use based on both offline and online testing (e.g. FFTW, Spiral, Custom FFT)

One CRAFFT feature
3-d FFT times using FFTW wisdom under-the-covers

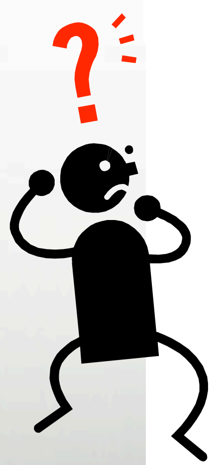
	128x128	256x256	512x512
FFTW plan	74	312	2758
FFTW exec	0.105	0.97	9.7
CRAFFT plan	0.00037	0.0009	0.00005
CRAFFT exec	0.139	1.2	11.4

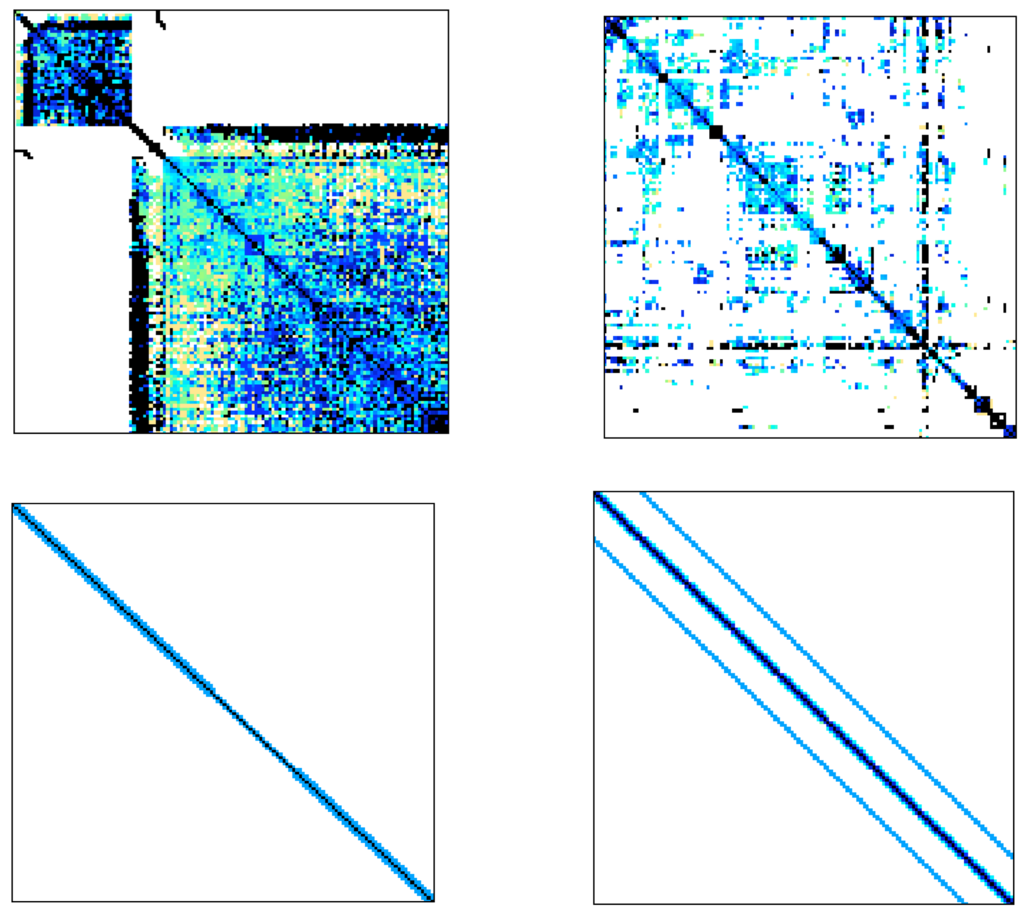
Iterative Solvers

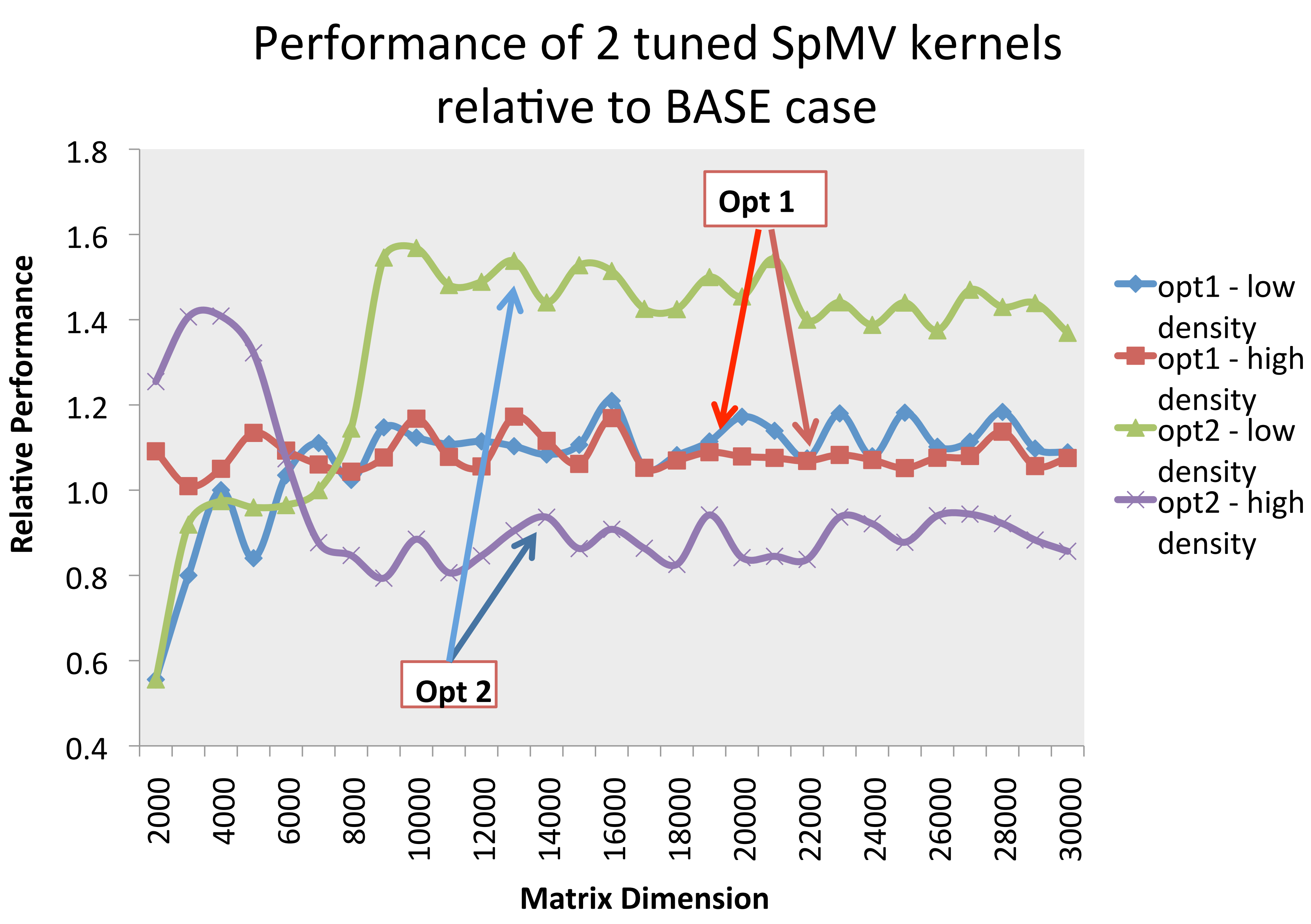
- Most important mathematical area for scaling HPC apps
- 25-75% of time in iterative solver is in the sparse MV kernel
- Unlike dense solvers, sparse solvers do not exhibit predictable performance with respect to different matrix types
  - Performance directly relates to sparsity characteristics
  - There is no such thing as a 'general purpose tuned kernel
  - What we require are kernels tuned for a specific matrix
  - Any compiler optimized code will remain useful only for a specific matrix category
  - Need to look at thousands of combinations of optimizations
- It is even worse than that…
  - We can make no reasonable assumptions about the interplay of optimizations with one another

Sparse Matrices

- Performance of SpMV depends on nonzero patterns of a matrix
- Nonzero patterns are application dependent
- Often determined on the fly

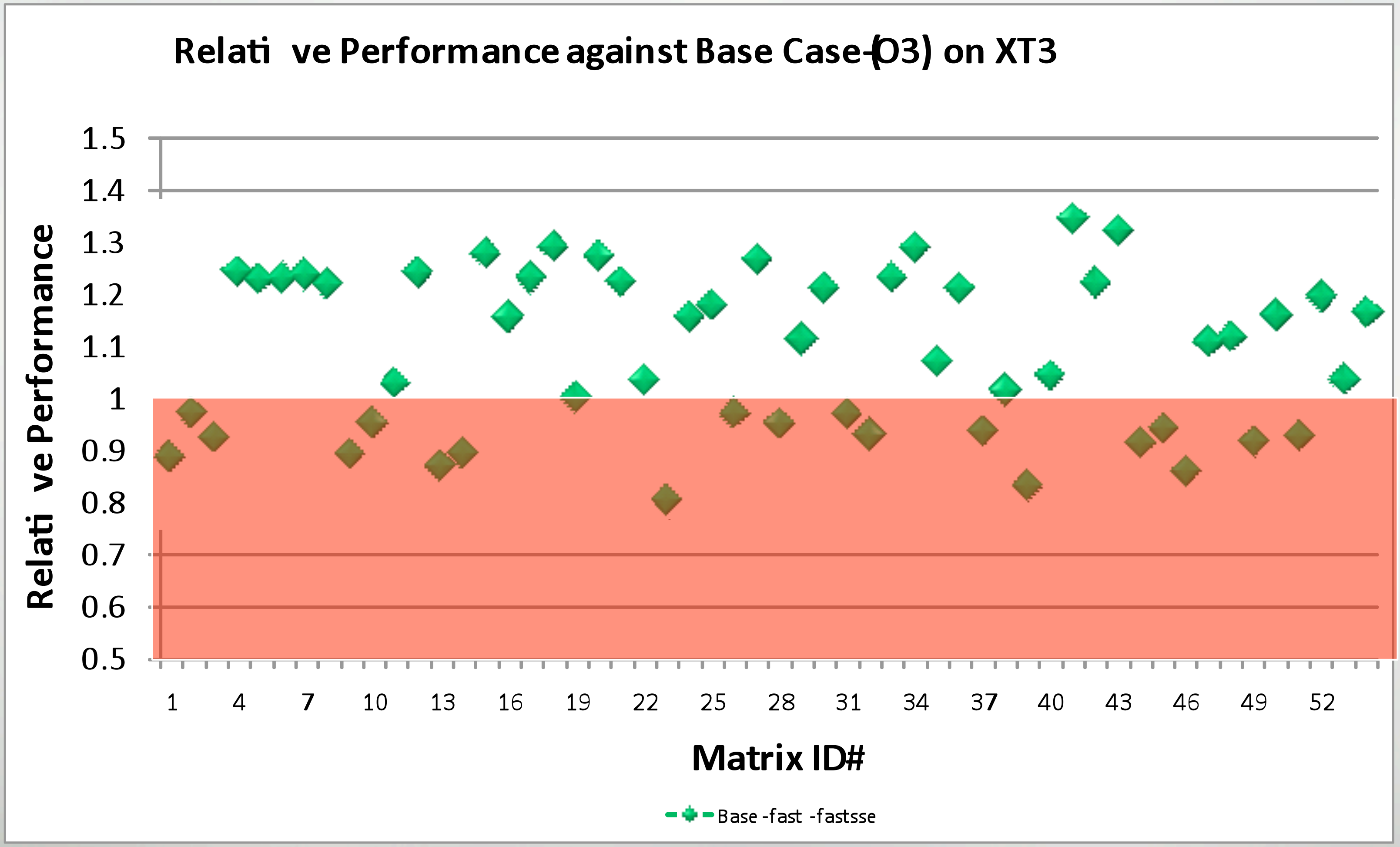






Performance of 2 tuned SpMV kernels relative to BASE case
Relative Performance
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
Opt 1
Opt 2
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
22000
24000
26000
28000
30000
Matrix Dimension
opt1 - low density
opt1 - high density
opt2 - low density
opt2 - high density

Same problem with aggressive tuning by compiler (PGI using -fast –fastsse)



Cray Adaptive Sparse Kernel (CASK)

- The CASK Process
  1. Analyze matrix at minimal cost
  2. Categorize matrix against internal classes
  3. Based on offline experience, find best CASK code for particular matrix class
  4. Previously assign “best” compiler flags to CASK code
  5. Assign best CASK kernel and perform Ax

- Goal is to have CASK silently sit beneath PETSc and Trilinos on Cray systems

- Released with libsci with PETSc 3.0.0
  - Generic and blocked CSR format (AIJ, BAIJ)
  - Support Triangular Solution for local IC/ILU preconditioning

Cray Adaptive Sparse Kernels (CASK) user perspective

Large-scale application

- Highly portable
- User controlled

(PETSc / Trilinos) / Hypre

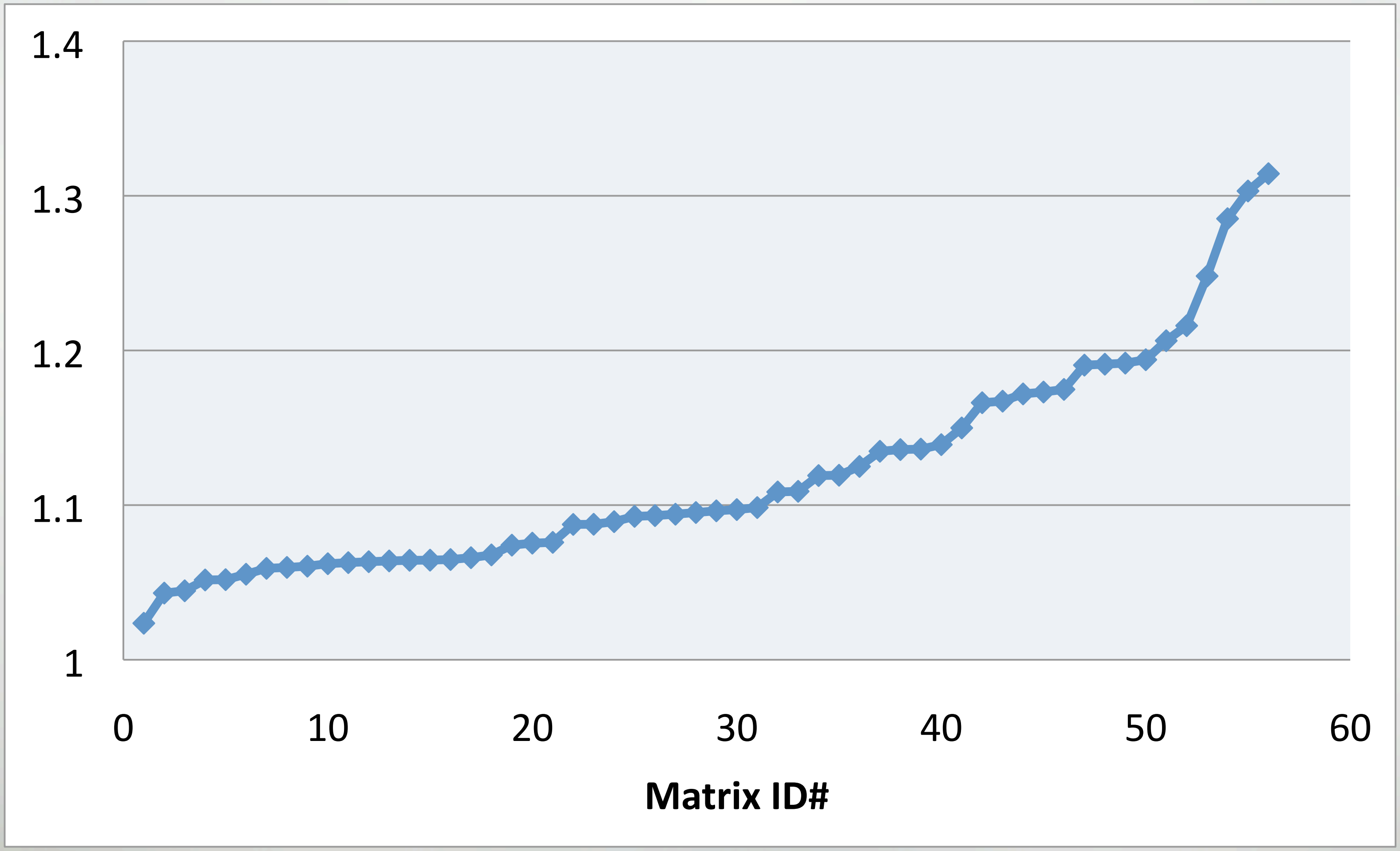
- Highly portable
- User controlled

CASK

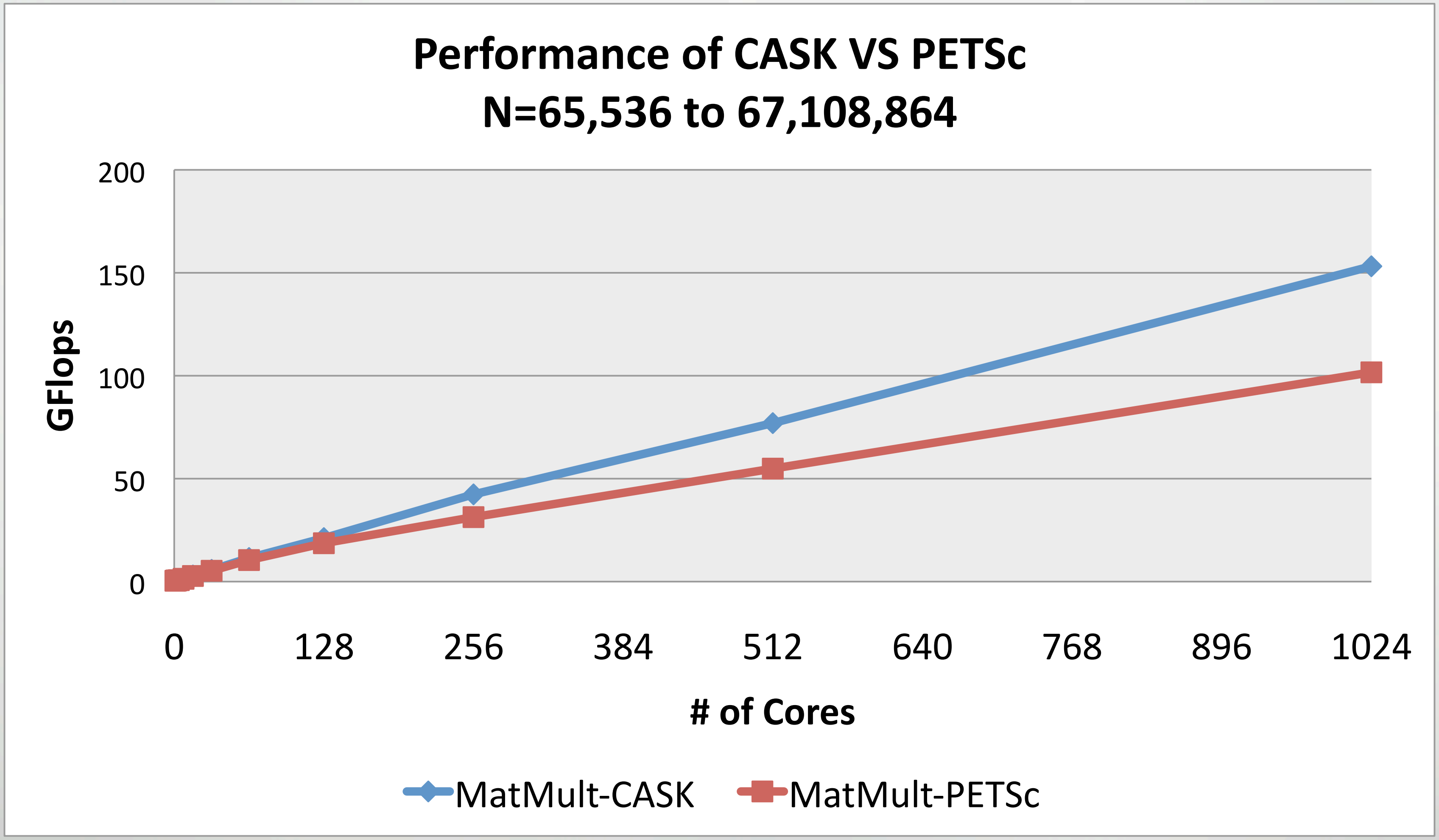
- Tuned for XT3/4/5
- User not aware

CASK and PETSc: Single Node XT5

Parallel SpMV on 8 cores

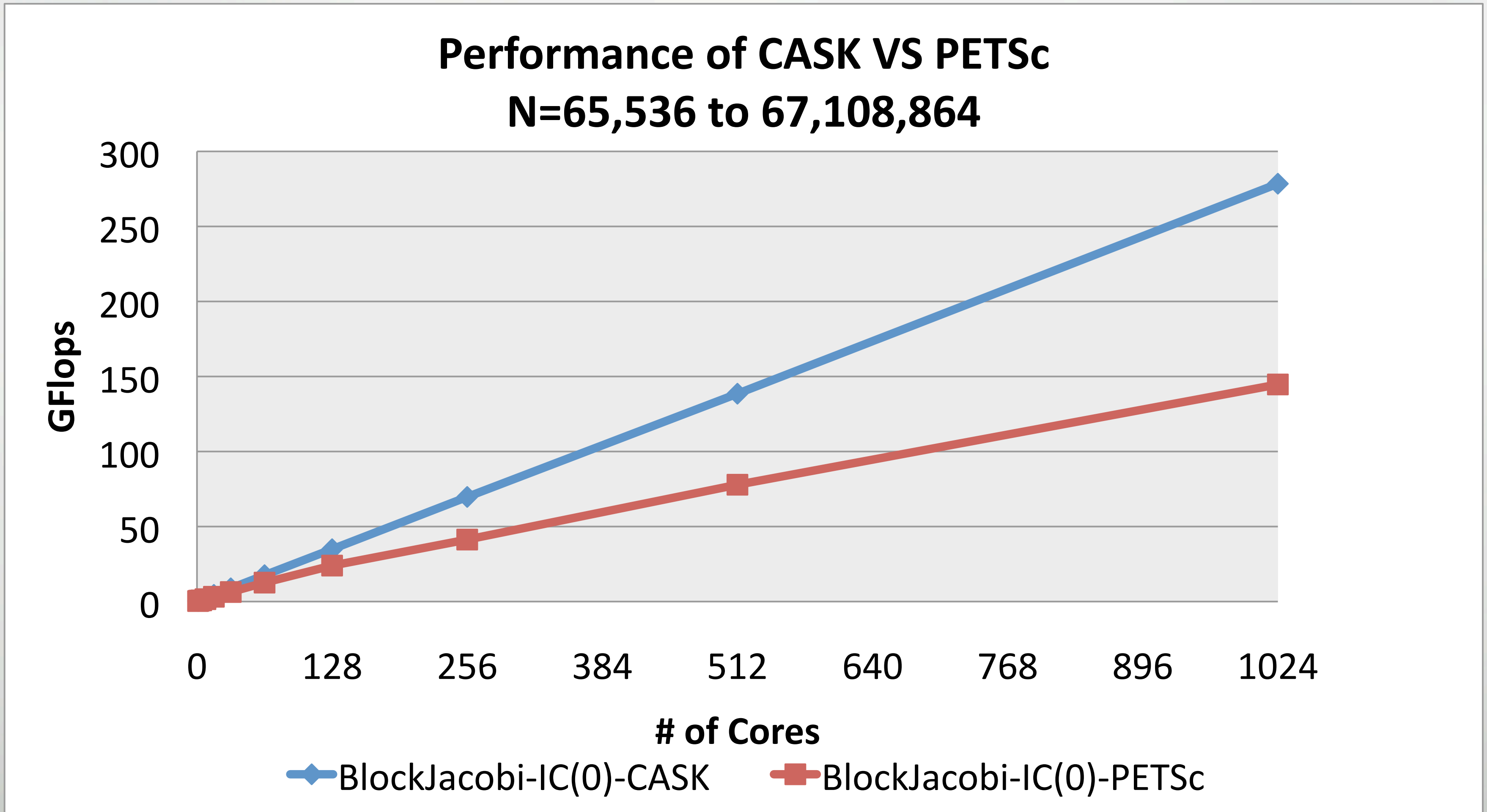


CASK Scalability: SpMV (XT4)



Setup time included

CASK Scalability: Block Jacobi Preconditioning (XT4)



More info?

- Thanks to the rest of Scientific Libraries team
- Send email to adrian@cray.com if you have any questions

THE SUPERCOMPUTER COMPANY